10/586800
IAP11 Rec'd PCT/PTO 19 JUL 2006

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

UPDATING ENTRIES CACHED BY A NETWORK PROCESSOR

Inventors: Yongxiang Han
Yu Zhang
Zhihong Yu

Prepared by: Jeffrey B. Huter
Patent Attorney

51 Cuppage Road
#06-01/02 StarHub Centre
Singapore 229469
Phone: (65) 62131481
Facsimile: (65) 62131018

"Express Mail" label number: EV705983676US

UPDATING ENTRIES CACHED BY A NETWORK PROCESSOR

BACKGROUND

[0001] A network communication system transmits information in packets from a transmitter to a receiver through one or more routers which route the packets between nodes within a network or between networks. The router may comprise one or more network processors to process and forward the packets to different destinations, and one or more external memories to store entries used by the network processors, such as node configuration data, packet queue and flow configuration data, etc.

[0002] The network processor may comprise a control plane to setup, configure and update the entries in the external memories, and a data plane having a plurality of microengines to process and forward the packets by utilizing the entries. Each of the microengines may have a local memory to store entries of the external memories that are frequently used. Once the control plane updates entries in the external memory, it may send a signal to the microengine(s) of the data plane that may cache or store the updated entries in its local memory. In response to the signal, the microengine(s) may flush all entries stored in the local memory to make them consistent with entries stored in the external memory.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The invention described herein is illustrated by way of example and not by way of limitation in the accompanying figures. For simplicity and clarity of illustration, elements illustrated in the figures are not necessarily drawn to scale. For example, the dimensions of some elements may be exaggerated relative to other elements for clarity.

Further, where considered appropriate, reference labels have been repeated among the figures to indicate corresponding or analogous elements.

[0004] Fig. 1 shows an embodiment of a network device.

[0005] Fig. 2 shows an embodiment of a network processor of the network device of Fig. 1.

[0006] Fig. 3 shows an embodiment of a method implemented by a control plane of the network processor depicted in FIG. 2.

[0007] Fig. 4 shows an embodiment of another method implemented by a microengine of the network processor depicted in FIG. 2.

[0008] Fig. 5 shows a data flow diagram of an embodiment for updating entries cached by the network processor depicted in FIG. 2.

DETAILED DESCRIPTION

[0009] The following description describes techniques for updating entries cached in a network processor. In the following description, numerous specific details such as logic implementations, pseudo-code, means to specify operands, resource partitioning/sharing/duplication implementations, types and interrelationships of system components, and logic partitioning/integration choices are set forth in order to provide a more thorough understanding of the current invention. However, the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full software instruction sequences have not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation.

[0010] References in the specification to "one embodiment", "an embodiment", "an example embodiment", etc., indicate that the embodiment described may include a particular feature, structure, or characteristic, but every embodiment may not necessarily include the particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to effect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described.

[0011] Embodiments of the invention may be implemented in hardware, firmware, software, or any combination thereof. Embodiments of the invention may also be implemented as instructions stored on a machine-readable medium, that may be read and executed by one or more processors. A machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computing device). For example, a machine-readable medium may include read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.) and others.

[0012] An embodiment of a network device 8 to route packets of a network communication system is shown in Fig. 1. The network device 8 may comprise a network interface 10, a framer 11, one or more network processors 12/13, a switch fabric 14, and one or more external memories 15/16. Examples for the network device 8

may comprise an ATM switch (Asynchronous Transfer Mode), an IP router (Internet Protocol), a SDH DXC (Synchronous Digital Hierarchy Data-cross Connection), and the like.

[0013] The framer 11 may perform operations on frames. In an embodiment, the framer 11 may receive a line datagram from a network interface 10 of the network communication system, delimitate frames and extract payload, such as Ethernet packet from the frames. In another embodiment, the framer 11 may receive packets from network processor 13, encapsulate the packets into frames and map the frames onto the network interface 10. The framer 11 may further perform operations such as error detection and/or correction. Examples for the framer 11 may comprise a POS (packet over Synchronous Optic Network) framer, a High-Level Data Link (HDLC) framer or the like.

[0014] The network processors 12 and 13 may perform operations on packets. In an embodiment, the network processor 12 may process and forward the packets from the framer 11 to an appropriate port of another network device through the switch fabric 14. For example, the network processor 12 may assemble IPv4 (Internet Protocol version 4) packets into CSIX (Common Switch Interface Specification) packets, modify packet headers and payloads, determine appropriate ports and forward the CSIX packets to the appropriate ports of the another network device. The network processor 13 may process and forward packets from the switch fabric 14 to appropriate ports 20 of the network interface 10 through the framer 11. For example, the network processor 13 may reassemble CSIX packets into IPv4 packets, modify packet headers and payloads, determine appropriate ports 20 and forward the IPv4 datagrams to the appropriate ports

20. Examples for the network processors 12 and 13 may comprise Intel® IXP 2XXX (e.g., IXP2400, IXP2800) network processors.

[0015] The switch fabric 14 may receive and send packets from/to a network processor connected therewith. Examples for the switch fabric 14 may comprise a switch fabric conforming to CSIX or other fabric technologies such as HyperTransport, Infiniband, PCI-X, Packet-Over-Synchronous Optical Network, RapidIO, and Utopia.

[0016] The external memories 15 and 16 may store entries 155/165 used by the network processors 12 and 13 to process and forward the packets. The entries may comprise node configuration data, queue configuration data, flow configuration data, network routing data, etc. The external memories 15 and 16 may further buffer the packets. In one embodiment, the external memory 15/16 may comprise SDRAM (Synchronous Dynamic Random Access memory) to store packets and QDR SRAM (Quad Data Rate Static Random Access Memory) to store entries.

[0017] Other embodiments may implement other modifications and variations on the structure of the network device as depicted in Fig. 1. For instance, the network processors 12 and 13 may perform framing duties instead of the framer 11 and the switch fabric may be omitted in a single-box scenario. For another instance, the network processors 12 and 13 may be integrated as one.

[0018] An embodiment of the network processor 12 (or network processor 13) is shown in Fig. 2. As shown, the network processor 12 may comprise a control plane 211, a data plane 212 and a scratch pad 213 that are communicable with each other through a bus connection.

[0019] The control plane 211 may be implemented as an integrated circuit (IC) with one or more processing cores 214_1 ... 214_M such Intel® XScale® processing cores or StrongARM® processing cores to execute instructions to perform various tasks. In an embodiment, the processing cores 214_1 ... 214_M of the control plane 211 may execute instructions to setup, configure and update entries 155/165 stored in the external memories 15/16. For instance, the processing cores 214_1 ... 214_M may update the external memories 15/16 which contain entries such as, for example, configuration data for nodes, configuration data for each packet queue, configuration data for each packet flow, etc. In another embodiment, the processing cores 214_1 ... 214_M may further handle packets containing protocol message and routing information that may need relatively complex computations. For instance, the processing cores 214_1 ... 214_M may process routing protocol packets containing routing information such as, for example, RIP (Routing Information Protocol) packets, OSPF (Open Shortest Path First) packets, and the like.

[0020] The data plane 212 may comprise a plurality of microengines 215_1 ... 215_N in Fig. 2 that may be communicable with each other. Each of the microengines may comprise a plurality of threads 216_1 ... 216_K to process and forward packets and one or more local memories 218_1 ... 218_N to store instruction code 220 and entries 224. The local memory 218_1 ... 218_N may comprise a control store, a memory, general purpose registers, transfer registers, and/or other storage mechanisms. In an embodiment, the local memories 218_1 ... 218_N may comprise instruction code 220 executable by the threads 216_1 ... 216_K and one or more entries 224 consistent with the entries 155/165 of the external memories 15/16. The threads 216_1 ... 216_K may access the local memories

218_1 ... 218_N to fetch some useful information for packet forwarding. Entries 155/165 may be cached from the external memory 15/16 to the local memories 218_1 ... 218_N of the microengines 215_1 ... 215_N based upon some criteria, for example, whether the entries 155/165 are frequently used by one or more microengines 215_1 ... 215_N of the data plane 212. Further, the entries 155/165 cached by one microengine 215_1 ... 215_N may be different from the entries 155/165 cached by another microengine 215_1 ... 215_N.

[0021] The scratch pad 213 is accessible by both the processing cores 214_1 ... 214_M of the control plane 211 and the microengines 215_1 ... 215_N of the data plane 212. The scratch pad 213 may comprise a buffer 226_1 ... 226_N to store data for each microengines 215_1 ... 215_N. The buffers 226_1 ... 226_N may be implemented using various structures such as, for example, ring buffers, link lists, stacks, etc. In other embodiments, the scratch pad 215 may be regarded as a flat memory.

[0022] In an embodiment, processing cores 214_1 ... 214_M of the control plane 211 may update one or more entries 155/165 in the external memories 15/16 by adding, deleting or changing one or more entries 155/165, and may write information related to the updated entries 155/165 to each buffer 226_1 ... 226_N of the scratch pad 213 associated with a microengine 215_1 ... 251_N that stores the updated entries 155/165 in its local memory 218_1 ... 218_N. Then, the microengines 215_1 ... 215_N may extract information from its buffer 226_1 ... 226_N, read the updated entries 155/165 from the external memories 15/16 and update the corresponding entries 224 in the local memories 218_1 ... 218_N. The information written in the buffers 226_1 ... 226_N may comprise entry identifiers (e.g. addresses, entry numbers, entry pointers) that uniquely identify entries 155/165 of the external memories 15/16.

[0023] Other embodiments may implement other modifications and variations on the structure of the network processor as depicted in Fig. 2. For example, the network processor 12 may further comprise a hash engine, a peripheral component interconnect (PCI) bus interface for communicating, etc.

[0024] Fig. 3 shows a process implemented by one or more processing cores 214_1 ... 214_M of the control plane 211 to update an external entry 155/165 stored in an external memory 15/16. As shown, in block 301, the control plane 211 may update an entry 155 in the external memory 15. Then, in block 302, the control plane 211 may search for microengine(s) 215_1 ... 215_N of the data plane 212 affected by the updated entry 155. In one embodiment, the control plane 211 determines a microengine 215_1 ... 215_N is affected by the updated entry 155 by determining that the microengine 215_1 ... 215_N has the updated entry 155 cached in its corresponding local memory 218_1 ... 218_N.

[0025] The control plane 211 may implement block 302 in various ways. In an embodiment, the control plane 211 may determine the affected microengines 215_1 ... 215_N by referring to a table of the external memory 15/16 or scratch pad 213 that lists the microengines 215_1 ... 215_N having cached a particular external entry 155. For example, the control plane 211 may supply a CAM (content addressable memory) of the external memory 15 with an identifier (e.g. an address, index, hash value, etc.) for the updated entry 155 to obtain a list of microengines 215_1 ... 215_N that have the entry 155 cached. In particular, the CAM may return a data word having at least N bits wherein each bit indicates whether a corresponding microengine 215_1 ... 215_N has the updated entry 155 cached. However, it should be appreciated that the control plane 211 may utilize other techniques and structures to maintain a corresponds between entries

155/165 and the microengines 215_1 ... 215_N that have stored local copies of the entries 155/165.

[0026] In block 303, the control plane 211 may write information associated with the external entry 155 updated in block 301 to buffers 226_1 ... 226_N of microengines 215_1 ... 215_N affected by the updated entry 155. The information may comprise identifiers that identify external entry 155/165 that have been updated by the control plane 211.

[0027] For instance, if an entry 155 of external table 151 is updated in block 301, the control plane 211 may search for the microengines 215_1 ... 215_N. that store the entry 155 in their local memories 218_1 ... 218_N (block 302). Then, the control plane 211 in block 303 may write an identifier (e.g. address, entry number, entry pointer, and/or other data) for the updated entry 155 to the buffers 226_1 ... 226_N of the affected microengines 215_1 ... 215_N identified in block 302. For example, if the control plane 211 determines in block 302 that microengines 215_1 and 215_N have cached the updated entry 155, then the control plane 211 in block 303 may write an identifier for the entry 155 to the corresponding buffers 226_1 and 226_N to inform the microengines 215_1 and 215_N that the identified entry 155 has been updated.

[0028] In another embodiment, if all entries or more than a threshold level of entries of the external memory 15 are updated in block 301, the control plane 211 may forgo block 302 and write a wildcard identifier to the buffers 226_1 ... 226_N indicating all cached entries 155 of the external memory 15 are invalid or outdated.

[0029] Fig. 4 shows an embodiment of a method to update one ore more entries 224 of local memories 218_1 ... 218_N of the data plane microengines 215_1 ... 215_N. In block 402, one thread 216_1 ... 216_K of each microengine 215_1 ... 215_N of the data plane 212

may be designated or otherwise configured to perform the task of updating cached entries 224 of the microengine 215_1 ... 215_N. In one embodiment, the control plane 211 may designate a thread 216_1 ... 216_K of each microengine 215_1 ... 215_N that is to update the cached entries 224 of the microengine 215_1 ... 215_N. Other embodiments may utilize other techniques to designate the thread to update the cached entries 224. For example, the microengine 215_1 ... 215_N may designate the thread, the thread may be predetermined by the instruction code 220, and/or the thread designation may be hardwired into the microengine 215_1 ... 215_N. In block 404, a thread 216_1 ... 216_K of a microengine 215_1 ... 215_N may be selected to continue executing its assigned tasks. To this end, the microengine 215_1 ... 215_N and/or the control plane 211 may awaken and/or otherwise activate the selected thread using various thread scheduling algorithms such as, for example, round robin, priority, weighted priority, and/or other scheduling algorithms.

[0030] In block 406, the selected thread 216_1 ... 216_K may determine whether the selected thread 216_1 ... 216_K is designated to update the local memory 218_1 ... 218_N of its microengine 215_1 ... 215_N. If selected thread 216_1 ... 216_K determines in block 406 that another thread 216_1 ... 216_K is designated for updates, then the selected thread 216_1 ... 216_K in block 408 may continue to process packets in a normal fashion. If, however, the selected thread 216_1 ... 216_K is designated to update its local memory 218_1 ... 218_N, then the thread 216_1 ... 216_K in block 410 may determine whether the buffer 226_1 ... 226_K for its microengine 215_1 ... 215_N indicates that entries 226 are invalid or outdated.

[0031] The selected thread 216_1 ... 216_K may implement block 410 in various ways. For an embodiment wherein the buffers 226_1 ... 226_N are scratch rings, the selected thread 216_1 ... 216_K may execute a predetermined instruction (e.g. 'br_!inp_state[...]') of the instruction code 220 and may determine whether a returned value of the predetermined instruction is true ('1') or false ('0'). The selected thread 216_1 ... 216_K may determine no updates are pending if the returned value is false, and likewise may determine one or more entries 226 of its local memory 218_1 ... 218_N are to be updated if the returned value is true.

[0032] If the selected thread 216_1 ... 216_K determines to update entries 224 of its local memory 218_1 ... 218_N, the thread 216_1 ... 216_K in block 412 may extract identifiers for the updated entries 155/165 from the buffer 226_1 ... 226_N associated with the microengine 215_1 ... 215_N of the thread 216_1 ... 216_K. The information may comprise an entry identifier that uniquely identifies the updated entries 155/165 of the external memories 15/16. Such an identifier may comprise an external memory number, an external memory pointer, an entry number, an entry pointer, and/or other identifying information from which an entry 155/165 may be discerned. However, if the selected thread 216_1 ... 216_K determines to update no entries 224 of its local memory 218_1 ... 218_N, the selected thread 216_1 ... 216_K may continue to block 408 to perform normal packet processing.

[0033] In block 414, the selected thread 216_1 ... 216_K may read entries 155/165 from the external memory 15/16 that have been identified by information in its corresponding buffer 226_1 ... 226_N as being updated. Further, the selected thread 216_1 ... 216_K may

update corresponding cached entries 224 based upon the entries read from the external memory 15/16 (block 416).

[0034] Other embodiments may implement other modifications and variations to the process as depicted in Fig. 4. For example, a microengine 215_1 ... 215_N may not assign a single thread 216_1 ... 216_K to perform the task of updating local memory 218_1 ... 218_N. Instead, each thread 216_1 ... 216_K of the microengine 215_1 ... 215_N may determine whether to update entries 224 cached in its local memory 218_1 ... 218_N before continuing with normal packet processing.

[0035] A data flow diagram illustrating an embodiment of updating entries 224 of local memories 218_1 ... 218_N of the network processor 12 is shown in Fig. 5. As shown, the control plane 211 may update one or more external entries 155 in an external memory 15 (arrow 501). Then, the control plane 211 may write information associated with the updated external entries 155 to the buffers 226_1 ... 226_N assigned to the affected microengines 215_1 ... 215_N (arrow 502). In response to a thread 216_1 ... 216_K determining, based upon information stored in its buffers 218_1 ... 218_N, that one or more cached entries 224 of its microengine 215_1 ... 215_N have been updated, the thread 216_1 ...216_K may read the updated external entries 155 from the external memory 15 (arrow 504) and update the corresponding local memory 218_1 ... 218_N with the read entries 155 (arrow 505).

[0036] While certain features of the invention have been described with reference to example embodiments, the description is not intended to be construed in a limiting sense. Various modifications of the example embodiments, as well as other

embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.